EECS 279: Approximaiton Algorithms NP-Completeness

Sungjin Im

University of California, Merced

09-02-2014

Significance of the question if $P \neq_? NP$

Perhaps you have heard of (some of) the following terms: NP, NPC, P, NP-hard, co-NP, EXP.

And the question if $P = NP$ or not.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.
- Problems in different complexity classes may have different challenges.

Significance of the question if $P \neq_? NP$

Why do you need to care about $P \neq_? NP$?

- A central question in computer science and mathematics.
- A lot of practical implications.
- Problems in different complexity classes may have different challenges.
- NP, NPC problems are widely found in practice.

Significance of the question if $P \neq_? NP$

The Millennium Prize Problems are seven problems in mathematics that were stated by the Clay Mathematics Institute in 2000. As of July 2013, six of the problems remain unsolved. A correct solution to any of the problems results in a US $1,000,000 prize (sometimes called a Millennium Prize) being awarded by the institute. The Poincare conjecture was solved by Grigori Perelman, but he declined the award in 2010.

The question $P \neq_? NP$ is the first in the list...

(source: wikipedia)

Polynomial time in what sense?

Class *P*

Definition

An algorithm for a problem is said to run in polynomial time, or said to be a polynomial-time algorithm, with respect to a particular model of computer (such as a RAM) if the number of instructions executed by the algorithm can be bounded by a polynomial in the size of the input.

RAM (Random Access Machine). Machine has infinite memory that allows random access.
Turing Machine. Often used in computational complexity.
RAM and TM are essentially "polynomially" equivalent. So let's focus on RAM.

Class P

Justification

P is invariant for all models of computation that are polynomially equivalent to the deterministic single-tape Turing Machine.

Example.

Theorem

Let $t(n)$ be a function, where $t(n) \geq n$. Then every $t(n)$ time multitape Turing machine has an equivalent $O(t^2(n))$ time single-tape Turing machine.

In fact, it can be shown that all reasonable deterministic computational models are polynomially equivalent.

Class P

x: instance
$|x|$: size of x.
The number of bits in the encoding of x.

We say that algorithm A is polynomial-time algorithm if there exists a polynomial $p(n)$ such that the running time of A is $O(p(|x|))$.

Decision vs. Optimization

Optimization version of TSP.
Input: a complete undirected graph $G = (V, E)$ with cost $c(e) \geq 0$ on each edge e.
Output: A Hamiltonian cycle with the minimum cost.

Decision version of TSP.
Input: a complete undirected graph $G = (V, E)$ with cost $c(e) \geq 0$ on each edge e, and a target L.
Output: Yes or no. Is there a Ham cycle with cost less than or equal to L?.

Yes instance. No instance.

Class *NP*

A decision problem is said to be in the problem class NP if there exists a verification algorithm $V(\cdot,\cdot)$, and two polynomials, p_1 and p_2, such that

1. for every "Yes" instance x of the problem, there exists a proof y with $|y|$ with $|y| \leq p_1(|x|)$. such that $V(x, y)$ outputs "Yes"'
2. for every "No" instance x of the problem, for all proofs y with $|y| \leq p_1(|x|)$, $V(x, y)$ outputs "No"'
3. the running time of $V(x, y)$ is $O(p_2(|x| + |y|)$.

Class *NP*

Is TSP in NP?

Is $\overline{\text{TSP}}$ in NP?

Class *NP*

Is TSP in NP?

Is $\overline{\text{TSP}}$ in NP?

Polynomial time reduction

Definition

Given two decision problems A and B, we say that there is a polynomial-tim reduction from A to B (or A reduces to B in polynomial time), denoted by $A \preceq B$, if there is a polynomial time algorithm that takes as input an instance of A and produces as output an instance of B and has the property that a "Yes" instance of B is output if and only if a "Yes" instance of A is input.

NP-hard and NP-complete

Definition
A problem B is NP-hard if for every problem A in NP reduces to B in polynomial time.

Definition
A problem B is NP-complete if B is in NP and NP-hard.

Theorem
Let B be an NP-complete problem. If B has a polynomial-time algorithm, then $P = NP$.

NP-hard and NP-complete

Theorem

Polynomial-time reductions are transitive, i.e. if $A \preceq B$ and $B \preceq C$, then $A \preceq C$.

Corollary

If A is in NP, B is NP-complete, and $B \preceq A$, then A is also NP-complete.

Gives an “easy” way to show a problem is NP-complete.

The “source” NP-complete problem.

A Boolean formula is an expression involving Boolean variables and operations. For example, $\Phi = (\bar{x} \wedge y) \vee (x \wedge \bar{z})$ is a boolean formula. We say a boolean formula is satisfiable if some assignment of 0s and 1s to the variables makes the formula evaluate to 1. Note that Φ is satisfialbe ($x = 0$, $y = 1$, $z = 0$).

Satisfiability problem (SAT)
$SAT = \{ < \Phi > \ | \ \Phi \text{ is a satisfiable Boolean formula}\}$

Theorem (Cook-Levin Theorem)

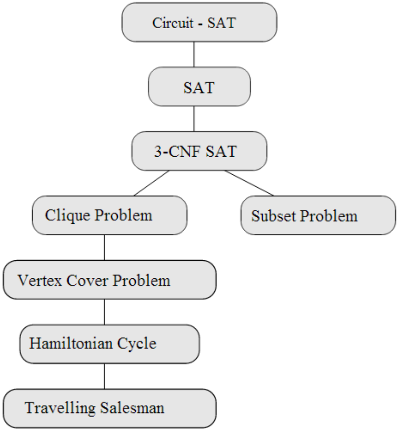
SAT is NP-complete.

NP-completeness

NPC derivation tree

Theorem

SAT is NP-complete.



Strongly weakly NP-complete

Definition

An NP-complete problem B is strongly NP-complete if it is NP-complete even when its numeric data are encoded in unary. It is said to to be weakly NP-complete if it has a pseudopolynomial time algorithm (that is, it has a polynomial-time algorithm if its numeric data are encoded in unary).

Strongly/weakly NP-complete
Examples

Partition Problem
Input: positive integers $a_1, a_2, ..., a_n$ such that $\sum_{i=1}^{n} a_i$ is even.
Goal: To decide if there is a partition of $\{1, 2, ..., n\}$ into S and T such that $\sum_{i \in S} a_i = \sum_{i \in T} a_i$.

Strongly/weakly NP-complete

Examples

Partition Problem
Input: positive integers $a_1, a_2, ..., a_n$ such that $\sum_{i=1}^{n} a_i$ is even.
Goal: To decide if there is a partition of $\{1, 2, ..., n\}$ into S and T such that $\sum_{i \in S} a_i = \sum_{i \in T} a_i$.
Partition Problem is weakly NP-hard. There is an algorithm with running time $O(n \sum_i a_i)$.

Strongly/weakly NP-complete

Examples

Partition Problem
Input: positive integers $a_1, a_2, ..., a_n$ such that $\sum_{i=1}^{n} a_i$ is even.
Goal: To decide if there is a partition of $\{1, 2, ..., n\}$ into S and T such that $\sum_{i \in S} a_i = \sum_{i \in T} a_i$.
Partition Problem is weakly NP-hard. There is an algorithm with running time $O(n \sum_i a_i)$.

3-Parition Problem
Input: positive integers $a_1, a_2, ..., a_{3n}$, b such that $b/4 < a_i < b/2$ for all i, and $\sum_{i=1}^{3n} a_i = nb$.
Goal: To decide if there is a partition of $\{1, 2, ..., 3n\}$ into n sets T_j such that $\sum_{i \in T_j} a_j = b$ for all $j = 1, 2, ..., n$.

Strongly/weakly NP-complete

Examples

Partition Problem
Input: positive integers $a_1, a_2, ..., a_n$ such that $\sum_{i=1}^{n} a_i$ is even.
Goal: To decide if there is a partition of $\{1, 2, ..., n\}$ into S and T such that $\sum_{i \in S} a_i = \sum_{i \in T} a_i$.
Partition Problem is weakly NP-hard. There is an algorithm with running time $O(n \sum_i a_i)$.

3-Parition Problem
Input: positive integers $a_1, a_2, ..., a_{3n}$, b such that $b/4 < a_i < b/2$ for all i, and $\sum_{i=1}^{3n} a_i = nb$.
Goal: To decide if there is a partition of $\{1, 2, ..., 3n\}$ into n sets T_j such that $\sum_{i \in T_j} a_j = b$ for all $j = 1, 2, ..., n$.
3-Partition is strongly NP-hard.